BEST AVAILABLE IMAGES

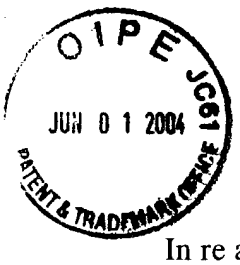
Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re application of: Joshi, et al.

Confirmation No. 9515

Group Art Unit No.: 2124

Serial No.: 09/717,187

Examiner: T.A. Vu

Filed: November 20, 2000

For: METHOD AND APPARATUS FOR DEBUGGING A SOFTWARE PROGRAM USING DYNAMIC DEBUG PATCHES AND COPY ON WRITE VIEWS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Sir:

This Appeal Brief is submitted in support of the Notice of Appeal filed on May 25, 2004.

I. REAL PARTY IN INTEREST

Oracle Corporation is the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF CLAIMS

Claims 1 - 24 are pending in this application, have been finally rejected and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

No amendments were filed after the final Office Action.

V. SUMMARY OF THE INVENTION

Like most software systems, a database server has complicated shared memory data structures that contain data and control information for a portion of a database system. Because of software, hardware or firmware bugs that may exist in a database system, shared memory structures may become logically incorrect and cause a database system to fail. For example, attempting to process corrupted pointers often causes a database system to fail, after which normal database operations are no longer possible. Database failures are typically discovered by checking consistency of data structures, by verifying certain assumptions, or by running into corrupted pointers.

The types of recovery operations performed after a failure vary depending on the type of failure that occurred, the structures affected and the type of recovery that is performed. If no files are lost or damaged, recovery may amount to no more than rebooting the database system. On the other hand, if data has been lost, additional steps may be required to recover the database. Once the database has been recovered, normal operations may continue. But because the root cause is still undetermined and therefore unresolved, the error condition may reoccur. Therefore, it is still important to diagnose the state of the data structures and data surrounding the database failure. Such a diagnosis may provide valuable information that can reduce the chance of future failures.

One problem with traditional debugging techniques is that they can be intrusive. For example, a database system that supports the Structured Query Language (SQL) may be debugged by compiling SQL statements and running them against the database. The act of compiling and executing SQL statements may change the state of the database system. As

another example, the use of debugging software may cause changes to data structures within the complex memory structures of database systems. Thus, although data structures are ideally left untouched upon a failure, diagnosing the failure may involve working directly on the same data structures from which data is to be obtained. The mere act of diagnosing the problem can easily make the problem worse because diagnosis may involve altering the state of the database. Further, where debug operations are performed on the database while the database is down, multiple programmers cannot each privately diagnose the failure, since each programmer's work would interfere with the concurrent debugging progress of fellow programmers. As a result, data structures of interest are typically diagnosed by a single programmer manually entering debug commands into a console. Hence, it is difficult to both be non-intrusive on a database and at the same time obtain a sufficient amount of meaningful data for debugging (Specification, Pages 2-5).

The inventions recited in Claims 1-24 address the problem of how to allow sufficient information about failures to be obtained during debugging of computer software while addressing issues related to over intrusive approaches. This problem is addressed by an approach for allowing multiple persons to concurrently test software patches on a software program or debug a problem of the software program. Each person is provided with their own private view that includes both copied portions of the software program that reflect modifications made by that person and the portions of the preserved software program that the person has not modified. Providing a private view to each person allows each person to test and debug privately, independently, and concurrently with others. Each private view may be extensively explored and modified without affecting the memory state of the software program that existed at the time the software program was shutdown.

Accordingly, at any time, each private view may be refreshed to the state of the software program that existed at the time of shutdown. Faster diagnosis of the problem may therefore be accomplished because a debugger does not have to peek cautiously and slowly into the inner-workings of the software program. Similarly, the testing of various potential

solutions to bugs in the program may be accomplished efficiently without affecting the memory state of the software program that existed at the time the software program was shutdown. Thus, where downtime of a software program must be kept to a minimum, the present techniques allow for performing quick and comprehensive diagnostics and testing of potential solutions to problems in the software program (Specification, Page 6).

FIG. 1 illustrates how data is copied to preserve the memory state of a software program before being modified. For example, in FIG. 1, in response to compiling, dynamically linking and executing software program 1, which is a secondary software program, a user 1 modifies 106a and 106b is created. In response to compiling, dynamically linking and executing software program 2, which is another secondary software program that is distinct from software program 1, a user 2 modifies 106a and 106c is created. User 1 is unable to see the changes in 106c, and user 2 is unable to see the changes in 106b. The programmer of a particular debug and testing session has a private view of the original software program, which is being debugged, in a privately modified state. This privately modified state is a side effect of the aforementioned copying technique. In the case of a secondary software program that is a diagnostic tool, extensive exploration of the privately viewed data has the benefit of extracting valuable state information that may indicate the source of the problem. The privately viewed data may be extensively explored and modified without fear of altering the preserved portion 102, which represents the original data. In the case of a secondary software program that is a potential solution to a problem in the original software program, testing of the potential solution may be safely performed before publishing and or permanently adopting the potential solution. (FIG 1; Specification, Page 8, lines 15-20; Page 12, lines 7-13 and Page 13, lines 1-10).

VI. ISSUE

Whether Claims 1-24 are unpatentable under 35 U.S.C. § 103(a) over *Gorshkov et al.*, U.S. Patent No. 6,490,721 (hereinafter "*Gorshkov*") in view of *Bowman-Amuah*, U.S. Patent No. 6,442,748 (hereinafter "*Bowman*") and further in view of *Khoyi* et al., U.S. Patent No. 5,303,379 (hereinafter "*Khoyi*").

VII. GROUPING OF CLAIMS

Separate claim groups are not asserted.

VIII. ARGUMENTS

A. Introduction

In the final Office Action dated April 8, 2004, Claims 1-24 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Gorshkov et al.*, U.S. Patent No. 6,490,721 (hereinafter "*Gorshkov*") in view of *Bowman-Amuah*, U.S. Patent No. 6,442,748 (hereinafter "*Bowman*") and further in view of *Khoyi* et al., U.S. Patent No. 5,303,379 (hereinafter "*Khoyi*"). It is respectfully submitted that Claims 1-24 are patentable over *Gorshkov*, *Bowman* and *Khoyi* for at least the reasons provided hereinafter.

It is well founded that to establish a *prima facie* case of obviousness under 35 U.S.C. § 103(a), the references cited and relied upon must teach or suggest all the claim limitations. In addition, a sufficient factual basis to support the obviousness rejection must be proffered. *In re Freed*, 165 USPQ 570 (CCPA 1970); *In re Warner*, 154 USPQ 173 (CCPA 1967); *In re Lunsford*, 148 USPQ 721 (CCPA 1966). With respect to the present application, it is respectfully submitted that *Gorshkov*, *Bowman* and *Khoyi*, considered alone or in combination, do not in any way teach or suggest all the limitations of Claims 1-24. It is further submitted that a sufficient factual basis has

not been proffered in the final Office Action mailed April 8, 2004 to support the rejection of Claims 1-24 under 35 U.S.C. § 103 as being unpatentable over *Gorshkov* in view of *Bowman* and further in view of *Khoyi*.

B. Claims 1-24 Recite One or More Limitations That Are Not in Any Way Taught or Suggested by *Gorshkov*, *Bowman* and *Khoyi*

CLAIM 1

It is respectfully submitted that *Gorshkov*, *Bowman* and *Khoyi*, considered alone or in combination, do not in any way teach or suggest the Claim 1 limitation of "if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program."

Gorshkov is the primary reference relied upon by the Examiner in the prosecution of the present patent application. *Gorshkov* describes a software debugging approach that uses dynamic linking to link new user actions into an existing executable image to provide debugging functionality. The modified executable image is then executed and the new user actions are executed. The *Gorshkov* approach is described in the context of a parent process and a child process. According to *Gorshkov*, the parent process consists of the target program to be debugged and the debugging user actions needed from the user action libraries (Col. 3, lines 45-53). The child process is created using a fork process and is a copy of the parent process. In operation, the child process attaches to and patches into the parent process by inserting calls to debugging routines into the parent process. More specifically, the child process inserts, into the memory image of the target program contained in the parent process, calls to user actions contained in the user action library. The child process allocates space for the patch in a patch

area in the parent process and replaces one or more instructions in the target program with a branch instruction to the patch area. The child process also generates code in the patch area to call the user actions, i.e., the debugging routines (Col. 4, lines 1-38). When the modified image of the target program contained in the parent process is executed, it calls the debugging routines contained in the user action library.

It is understood from the prosecution history that the Examiner considers the parent process described in *Gorshkov* to be the "first software program" recited in Claim 1 and the child process to be the "second software program" recited in Claim 1 (See, e.g., second Office Action, Page 3 "the linking of the child to the target user program being a copy of the parent is equivalent to linking second program to first"). It is also understood from the prosecution history that the Examiner considers the forking of the child process to be the "preserving a memory state of a preserved portion of the first software program" limitation recited in Claim 1 (See third Office Action, Page 2, "forking a child is equivalent to preserving a parent program, or first program").

During the prosecution of the present application, the Examiner has taken two different, but equally untenable, positions with respect to the claim term "targeted data" and both positions are discussed herein for purposes of completeness. In the first position set forth in the second (non-final) Office Action mailed on December 5, 2003, the Examiner asserted that the instructions in the parent process that are replaced with a branch instruction to the patch area are the "targeted data" recited in Claim 1 ("the to-be-modified areas of parent code being duplicated in child is equivalent to making copy of targeted area"; second Office Action, Page 3, 3rd paragraph). The Examiner also admitted that *Gorshkov* does not teach or suggest "making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program." Applicant fully agrees with this admission given that *Gorshkov* both does not teach or suggest modifying the child process and explicitly teaches modifying the parent process when the child

process patches the debugging routines into the parent process (Col. 2, lines 38-64; Col. 4, lines 29-38).

In the second position set forth in the third (final) Office Action mailed on April 8, 2004, the Examiner changed his assertion with respect to what data in *Gorshkov* is considered to be the "targeted data" recited in Claim 1. Instead of the instructions in the parent process that are replaced with a branch instruction to the patch area being the "targeted data," the targeted data "would be any data in the first software program that require[s] modification or reconciliation in the light of existing changes effected by the second software" (third Office Action, Page 10, lines 19-20). The Examiner goes on to state "[t]he rejection thereby addresses only the limitation that a copy of the preserved portion of the first software is made and modification to the execution state or control state of such copy (see FIG. 4) is effected without affecting the preserved portion of the parent or first software, and *Gorshkov* has met such limitation" (third Office Action, Page 10, line 20 through Page 11, line 2).

One problem with this position is that the Examiner does not identify any particular data in *Gorshkov* that satisfies the requirements of Claim 1. Claim 1 requires that the targeted data be located "in the preserved portion of the first software program." Claim 1 further requires that the targeted data must be data in the preserved portion of the first software program that would otherwise, i.e., normally, be modified by execution of the second software program. Instead of the targeted data being modified, a copy of the targeted data is made and modified to generate a modified copy of the targeted data. The Examiner relies upon mere speculation that data that satisfies the requirements of the targeted data of Claim 1 might exist in the debugging apparatus of *Gorshkov* and even admits "[h]ence, it is unclear as to which data in the parent process as shown by *Gorshkov* would be called targeted data and thus susceptible for modification" (third Office Action, Page 10, lines 17-18). In the absence of at least some teaching or suggestion in *Gorshkov* of some data that satisfies all of the requirements of the targeted data recited in Claim 1, it is respectfully submitted that the Examiner has not proffered a sufficient factual basis to support the assertion that *Gorshkov* teaches or suggests the limitation of "if execution of the second software program

would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program."

Furthermore, the Examiner's position is self contradictory from the standpoint that the Examiner has asserted that the child process contains both the "preserved portion of the first software program" with the targeted data that is not modified (See third Office Action, Page 2, "forking a child is equivalent to preserving a parent program, or first program") and the copy of the targeted data that is modified (See third Office Action, Pages 10-11, modifying child process). Claim 1 requires that "if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program," then the targeted data in the preserved portion of the first software program is not modified. Instead, a copy of the targeted data is made and the copy is modified "without modifying the targeted data that is in the preserved portion of the first software program." The Examiner has asserted that data in the parent process is preserved as recited in Claim 1 when the child process is created. The Examiner further asserts that the copy of the parent process in the child process is then updated to satisfy the "modifying the copy of the targeted data to generate a modified copy of the targeted data" limitation of Claim 1. There is no teaching or suggestion in *Gorshkov* that the child process is modified in this manner. Furthermore, the Examiner has not explained how the child process contains both the targeted data that is preserved and not modified and the copy of the targeted data that is modified. In view of the foregoing, it is respectfully submitted that the Claim 1 limitation of "if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program" is not in any way taught or suggested by *Gorshkov*.

We turn now to the other two references relied upon by the Examiner in the rejection of Claim 1, namely, *Bowman* and *Khoyi*. The Office Action refers to portions of *Bowman* (Col. 177, lines 38-58; Col. 293, lines 32-56; and FIGS. 177, 178) and *Khoyi* (Col. 3, lines 33-44 and Col. 38, lines 7-22) for their teachings related to incorporating changes in computer software. The text at Col. 177, lines 38-58 of *Bowman* describes a software configuration management solution that allows developers to make changes to an open edition or checked out copy of a class to maintain version control while increasing development bandwidth. The text at Col. 293, lines 32-56 of *Bowman* describes maintaining consistency of data by assigning each logical unit of work independent copies of a portion of a business model to prevent transactions from interfering with each other. The text at Col. 3, lines 33-44 of *Khoyi* describes an approach for creating new objects by making copies of a prototype copy of an object. Users may then modify the copies of the prototype copy of an object without affecting the prototype copy of the object. The text at Col. 38, lines 7-22 of *Khoyi* describes making copies of linked objects where in some situations a link is copied and in other situations linked data is copied. Also, copies of objects may be modified independently of the original version.

Thus, *Bowman* and *Khoyi* describe making copies of shared data, distributing the copies of the shared data and allowing the copies of the shared data to be updated independently of the original shared data. *Bowman* and *Khoyi*, however, do not in any way teach or suggest the specific steps recited in Claim 1. Furthermore, the Examiner has not asserted that *Bowman* and *Khoyi* teach or suggest the specific limitations recited in Claim 1. For example, *Bowman* and *Khoyi* do not teach or suggest "preserving a memory state of a preserved portion of the first software program," "dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program," "executing the second software program," and then managing the targeted data as recited in Claim 1. Specifically, "if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of

the targeted data without modifying the targeted data that is in the preserved portion of the first software program."

In view of the foregoing, it is respectfully submitted that Claim 1 includes one or more limitations that are not taught or suggested by *Gorshkov*, *Bowman* and *Khoyi*, taken alone or in combination, and is therefore patentable over *Gorshkov*, *Bowman* and *Khoyi*. It is further submitted that a sufficient factual basis has not been proffered by the Examiner to support the rejection of Claim 1 under 35 U.S.C. § 103 as being unpatentable over *Gorshkov* in view of *Bowman* and further in view of *Khoyi*.

CLAIMS 2-8

Claims 2-8 all depend from Claim 1 and include all of the limitations of Claim 1. It is therefore respectfully submitted that Claims 2-8 are patentable over *Gorshkov*, *Bowman* and *Khoyi* for at least the reasons set forth herein with respect to Claim 1. Furthermore, it is respectfully submitted that Claims 2-8 recite additional limitations that independently render them patentable over *Gorshkov*, *Bowman* and *Khoyi*.

CLAIMS 9-16

Claims 9-16 recite limitations similar to Claims 1-8, except in the context of computer-readable media. It is therefore respectfully submitted that Claims 9-16 are patentable over *Gorshkov*, *Bowman* and *Khoyi* for at least the reasons set forth herein with respect to Claims 1-8.

CLAIMS 17-24

Claims 17-24 recite limitations similar to Claims 1-8, except in the context of apparatuses. It is therefore respectfully submitted that Claims 17-24 are patentable over *Gorshkov*, *Bowman* and *Khoyi* for at least the reasons set forth herein with respect to Claims 1-8.

IX. CONCLUSION AND PRAYER FOR RELIEF

Based on the foregoing, it is respectfully submitted that the rejection of Claims 1-24 under 35 U.S.C. § 103 lacks the requisite factual and legal bases. Appellants therefore respectfully request that the Honorable Board reverse the rejection of Claims 1-24 under 35 U.S.C. § 103 over *Gorshkov*, *Bowman* and *Khoyi*.

Respectfully submitted,

HICKMAN PALERMO TRUONG & BECKER LLP

Date: May 27, 2004

Edward A. Becker
Registration No. 37,777

1600 Willow Street
San Jose, California 95125-5106
Tel: (408) 414-1204
Fax: (408) 414-1076

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: **Mail Stop Appeal Brief-Patents**, Commissioner for Patents, Washington, DC 20231

On May 27, 2004 By

CLAIMS APPENDIX

1. A method of debugging a first software program, the method comprising the steps of:
preserving a memory state of a preserved portion of the first software program;
dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program;
executing the second software program; and
if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program.

2. The method of Claim 1, further comprising the steps of:
publishing in the preserved portion of the first software program a corresponding symbolic name associated with the second software program; and
multiple users accessing the second software program is accessed through the corresponding symbolic name.

3. The method of Claim 1, wherein the first software program is a database system.

4. The method of Claim 1, wherein the step of preserving a memory state further includes the step of suspending a failed application of the database system.

5. The method of Claim 1, further including the step of, in response to a subsequent attempt to access the targeted data in the preserved portion of the first software program, accessing the modified copy of the targeted data.

6. The method of Claim 5, wherein the steps of dynamically linking and executing are initiated by a particular user, and wherein the step of accessing the modified copy occurs only if that particular user initiates the subsequent attempt to access the targeted data.

7. The method of Claim 1, wherein:

the steps of dynamically linking and executing the second software program are performed by a first user;

the modified copy is a first modified copy of the targeted data; and

the method further comprises the steps of:

after the first modified copy has been created for the first user, a second user executing performing an operation which, when executed, would cause modification to the targeted data in the preserved portion; and

performing the operation by making a second copy of the targeted data and modifying the second copy to generate a second modified copy of the targeted data, the second modified copy being separate from the first modified copy and from the preserved portion.

8. The method of Claim 7, further comprising the steps of:

after the first and second modified copies have been created for the first user and second user respectively, a third user dynamically linking and executing a third software program which, when executed, would cause modification to the targeted data in the preserved portion; and

making a third copy of the targeted data and modifying the third copy to generate a third modified copy, the third modified copy being separate from the first modified copy, from the second modified copy, and from the preserved portion.

9. A computer-readable medium bearing instructions for debugging a first software program, the instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

preserving a memory state of a preserved portion of the first software program;

dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program;

executing the second software program; and

if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program.

10. The computer-readable medium of Claim 9, further comprising the steps of:

publishing in the preserved portion of the first software program a corresponding symbolic name associated with the second software program; and

multiple users accessing the second software program is accessed through the corresponding symbolic name.

11. The computer-readable medium of Claim 9, wherein the first software program is a database system.

12. The computer-readable medium of Claim 9, wherein the step of preserving a memory state further includes the step of suspending a failed application of the database system.

13. The computer-readable medium of Claim 9, further including the step of, in response to a subsequent attempt to access the targeted data in the preserved portion of the first software program, accessing the modified copy of the targeted data.

14. The computer-readable medium of Claim 13, wherein the steps of dynamically linking and executing are initiated by a particular user, and wherein the step of accessing the modified copy occurs only if that particular user initiates the subsequent attempt to access the targeted data.

15. The computer-readable medium of Claim 9, wherein:

the steps of dynamically linking and executing the second software program are performed by a first user;

the modified copy is a first modified copy of the targeted data; and

the method further comprises the steps of:

after the first modified copy has been created for the first user, a second user executing performing an operation which, when executed, would cause modification to the targeted data in the preserved portion; and

performing the operation by making a second copy of the targeted data and modifying the second copy to generate a second modified copy of the targeted data, the second modified copy being separate from the first modified copy and from the preserved portion.

16. The computer-readable medium of Claim 15, further comprising the steps of:

after the first and second modified copies have been created for the first user and second user respectively, a third user dynamically linking and executing a third software program which, when executed, would cause modification to the targeted data in the preserved portion; and

making a third copy of the targeted data and modifying the third copy to generate a third modified copy, the third modified copy being separate from the first modified copy, from the second modified copy, and from the preserved portion.

17. An apparatus for debugging a first software program, wherein the apparatus comprises a memory storing one or more instructions which, when executed by one or more processors, cause the one or more processors to perform the steps of:

preserving a memory state of a preserved portion of the first software program;

dynamically linking a second software program to the first software program without deallocating from volatile memory the first software program;

executing the second software program; and

if execution of the second software program would otherwise cause modification to targeted data that is in the preserved portion of the first software program, then making a copy of the targeted data and modifying the copy of the targeted data to generate a modified copy of the targeted data without modifying the targeted data that is in the preserved portion of the first software program.

18. The apparatus of Claim 17, wherein the memory includes one or more additional instructions which, when executed by the one or more processors, cause the one or more processors to perform the additional steps of:

publishing in the preserved portion of the first software program a corresponding symbolic name associated with the second software program; and

multiple users accessing the second software program is accessed through the corresponding symbolic name.

19. The apparatus of Claim 17, wherein the first software program is a database system.

20. The apparatus of Claim 17, wherein the step of preserving a memory state further includes the step of suspending a failed application of the database system.

21. The apparatus of Claim 17, wherein the memory includes one or more additional instructions which, when executed by the one or more processors, cause the one or more processors to perform the additional step of, in response to a subsequent attempt to access the targeted data in the preserved portion of the first software program, accessing the modified copy of the targeted data.

22. The apparatus of Claim 21, wherein the steps of dynamically linking and executing are initiated by a particular user, and wherein the step of accessing the modified copy occurs only if that particular user initiates the subsequent attempt to access the targeted data.

23. The apparatus of Claim 17, wherein:

the steps of dynamically linking and executing the second software program are performed by a first user;

the modified copy is a first modified copy of the targeted data; and

wherein the memory includes one or more additional instructions which, when executed by the one or more processors, cause the one or more processors to perform the additional steps of:

after the first modified copy has been created for the first user, a second user executing performing an operation which, when executed, would cause modification to the targeted data in the preserved portion; and

performing the operation by making a second copy of the targeted data and modifying the second copy to generate a second modified copy of the targeted data, the second modified copy being separate from the first modified copy and from the preserved portion.

24. The apparatus of Claim 23, wherein the memory includes one or more additional instructions which, when executed by the one or more processors, cause the one or more processors to perform the additional steps of:

after the first and second modified copies have been created for the first user and second user respectively, a third user dynamically linking and executing a third software program which, when executed, would cause modification to the targeted data in the preserved portion; and

making a third copy of the targeted data and modifying the third copy to generate a third modified copy, the third modified copy being separate from the first modified copy, from the second modified copy, and from the preserved portion.